Московский государственный университет имени М. В. Ломоносова

Факультет Вычислительной Математики и Кибернетики

Кафедра Математических Методов Прогнозирования

ДИПЛОМНАЯ РАБОТА СТУДЕНТА 417 ГРУППЫ

«Улучшение автоматической стилизации изображений с помощью преобразующей нейросети»

Выполнил:

студент 4 курса 417 группы

Елистратов Семен Юрьевич

Научный руководитель:

к.ф-м.н., доцент

Китов Виктор Владимирович

Москва, 2022

Содержание

1 Введение **2**

2 Обзор **3**

2.1 Исходная модель 3

2.2 Conditional Instance Normalization 4

2.3 Модель с кодировщиком стилей 6

3 Улучшение модели с кодировщиком стилей **7**

3.1 Обучение модели 7

3.2 Контроль силы стилизации 8

3.3 Контроль силы перекраски 9

3.4 Дополнительные ручки контроля стилизации 10

4 Вычислительные эксперименты **11**

4.1 Сравнение результатов стилизации 12

4.2 Оценка контроля силы стилизации 13

4.3 Оценка контроля силы перекраски 14

4.4 Оценка контроля переноса стилевых паттернов 14

5 Заключение **16**

Список литературы **18**

1 Введение

Задача автоматического переноса стиля существовала еще до начала развития глубокого обучения. Рассматривались различные статистические методы, которые могли выделять стилевые паттерны и переносить их с одного изображения на другое. В основном такие методы основывались на вычислении различных статистик окрестностей пикселей. Использование методов глубокого обучения для решения задачи автоматического переноса стиля было впервые продемонстрировано в статье [1]. Авторы предложили крайне мощный и гибкий параметрический метод, превосходящий по качеству стилизации предыдущие подходы. Оптимизация производилась непосредственно в пространстве изображений, поэтому метод мог работать с произвольным стилем. Основной недостаток метода заключался в скорости его работы. Даже на мощных GPU стилизация занимала порядка нескольких минут. Для решения этой проблемы авторы статьи [2] предложили использовать отдельную сеть-преобразователь, которая обучается для заренее фиксированного стиля. Такой подход позволяет быстро выполнять стилизацию для любого изображения контента без потери качества. Однако сам процесс обучения данной сети достаточно долгий, а хранение для каждого стиля отдельной сети-преобразователя затратно по памяти. В статье [3] авторы заметили, что большинство весов нейросети инварианты относительно различных стилей. Они предложили подход, заключающийся в том, что только часть весов сети-преобразователя будет соответствовать конкретному стилю. Им удалось обучить сеть на некотором конечном наборе стилей и получить качественные стилизации для них. В дальнейшем была предложена мощная модель [5], которая обучалась на большом датасете стилей и позволяла получать качественные стилизации для любых стилей, даже не участвовавших в обучении. В моей работе было проведено исследование работы данного метода стилизации. Были выявлены недостатки модели и предложены модификации, позволяющие увеличить ее гибкость.

Параллельно разрабатывались методы улучшающие гибкость модели с сетью-преобразователем. В статье [6] был предложен подход, позволяющий пользователю на этапе тестрования контролировать силу стилизации. Для этого авторы использовали отдельную нейросеть, получавшую на вход параметр, отвечающий за силу

стилизации, и выдававшую значения некоторых весов сети-преобразователя. В статье [5] был предложен более простой подход для контроля стилизации, имеющий как достоинства, так и недостатки по сравнению с предыдущим методом.

В статье [7] была обозначена проблема изменения цветовой гаммы при стилизации, а также предложены методы решающие эту проблему. В своей работе я модифицировал один из методов путем добавления дополнительного параметра, контролирующего силу перекраски. Это позволяет производить более плавный контроль сохранения исходной цветовой гаммы контента.

2 Обзор

2.1 Исходная модель

Задача автоматического переноса стиля с помощью нейронных сетей заключается в следующем: на вход модели подаются два изображения: контент - I_c и стиль - I_s. Модель выдает новое изображение контента I_r в заданном стиле. Обычно изображение контента - некоторая фотография, а изображение стиля - картина, текстура или рисунок. В статье [1] авторы сформулировали следующие предположения:

- Два изображения схожи по контенту, если их высокоуровневые признаки, полученные с помощью некоторой модели распознавания, схожи в смысле евклидового расстояния.

- Два изображения схожи по стилю, если их низкоуровневые признаки, полученные с помощью некоторой модели распознавания, схожи в смысле евклидового расстояния между их пространственными статистиками.

В качестве модели распознавания в статье использовалась сеть VGG-16 [8]. На основе данных предположений авторы предложили использовать следующие функции потерь:

- Функция потерь контента:

$$\mathcal{L}_{content}(I_c, I_r) = \sum_{i \in L_{\mathcal{C}}} \frac{1}{n_i} \|\phi_i(I_r) - \phi_i(I_c)\|_F^2,$$

- Функция потерь стиля:

$$\mathcal{L}_{style}(I_s, I_r) = \sum_{i \in L_{\mathcal{S}}} \frac{1}{n_i} \|\mathcal{G}(\phi_i(I_r)) - \mathcal{G}(\phi_i(I_s))\|_F^2,$$

- Суммарная функция потерь

$$\mathcal{L}_{total}(I_c, I_s, I_r) = \mathcal{L}_{content}(I_c, I_r) + \lambda \mathcal{L}_{style}(I_s, I_r),$$

  где $L_{\mathcal{S}}$ and $L_{\mathcal{C}}$ - множество соответствущих слоев сети VGG, $\phi_i(\cdot)$ - карта признаков (выход соответствующего слоя), n_i - соответствующее число элементов тензора, λ - параметр отвечающий за силу стилизации.

В качестве пространственных статистик используются матрицы Грама - матрицы попарных скалярных произведений относительно каждого канала. Элемент этой матрицы с индексом cc' определется как:

$$\mathcal{G}(\phi_i(x))_{cc'} = \frac{1}{C_i H_i W_i} \sum_{h=1}^{H_i} \sum_{w=1}^{W_i} \phi_i(x)_{hwc} \phi_i(x)_{hwc'}$$

Тогда итоговое изображение вычисляется путем оптимизации в пространстве изображений:

$$I_r = \arg\min_x \mathcal{L}_{total}(I_c, I_s, x)$$

Так как такая оптимизация достаточно затратна по времени, было предложено использовать сеть преобразователь $T(\cdot)$ типа кодировщик-декодировщик, которая получает на вход изображение контента и выдает итоговое стилизованное изображение: $T(I_c) = I_r$ Модель обучается для фиксированного стиля на тех же функциях потерь. Обучается модель достаточно долго, однако быстро работает на тесте.

2.2 Conditional Instance Normalization

Обучать для каждого стиля свою сеть-преобразователь затратно по памяти. В статье [3] авторы утверждают, что только 0.2% всех весов сети имеют сильное различие для разных стилей. Они зафиксировали набор из 32 стилей и предложили модель, которая имеет одинаковые веса во всех блоках сети для разных стилей кроме параметров в условной нормализации (Conditional Instance Normalization):

$$\hat{x} = \gamma_s(\frac{x-\mu}{\sigma}) + \beta_s,$$

где μ и σ - поканальное среднее и стандартное отклонение соответственно.

Данная операция похожа на батч-нормализацию, однако, во-первых в данном случае среднее и стандартное отклонение вычисляется только по пространственным координатам, а во-вторых параметры γ_s и β_s фиксированы, а не являются весами модели, как в батч-нормализации. В данной модели именно параметры нормализации отвечают за конкретный фиксированный стиль. Выберем эту модель в качестве бейзлайна.

Смысл данной нормализации заключается в том, что внутреннее представление контента сначала нормализуется на свои пространственные статистики, а затем на параметры, которые хранят информацию о стиле. То есть именно эти параметры γ_s и β_s отвечают за перенос стиля.

Все же основной недостаток данной модели заключается в том, что она может работать только с конечным набором фиксированных стилей. Дальнейшие исследования были направлены на создание метода, которая в режиме тестирования может работать с произвольным стилем. В модели [4] предложен похожий подход. Сеть-преобразователь получает на вход изображение стиля, пропускает его через кодировщик и вычисляет те же пространственные статистики: среднее $\mu(I_s)$ и стандартное отклонение $\sigma(I_s)$. Затем сеть получает на вход изображение контента, которое после кодировщика идет на слой условной нормализации, где параметрами γ_s и β_s являются статистики $\mu(I_s)$ и $\sigma(I_s)$. Данный подход может работать с произвольным стилем и проводить быструю стилизацию, однако качество достаточно низкое из-за того, что в модели может использоваться лишь один блок условной нормализации. Такой метод нормализации с использованием пространственных статистик изображения стиля авторы назвали Adaptive Instance Normalization, а модель - AdaIn.

2.3 Модель с кодировщиком стилей

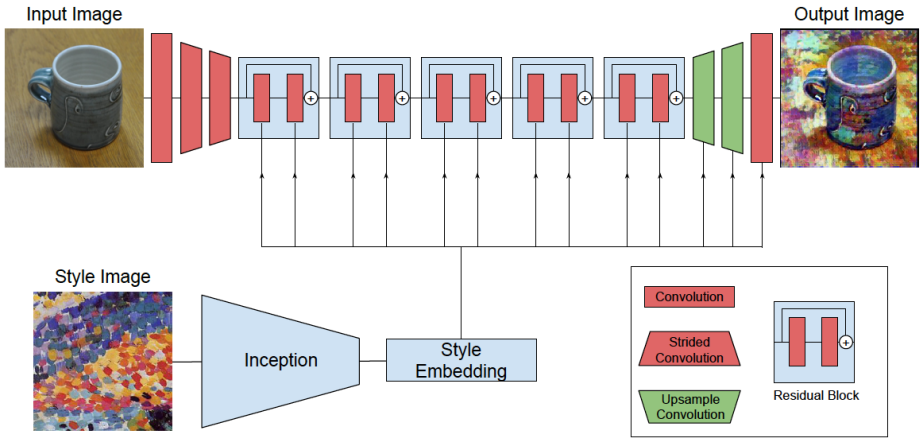


Рис. 1: Модель работающая с произвольным стилем. Сеть кодировщик стилей (внизу) получает векторное представление для изображения стиля. Сеть-преобразователь (вверху) получает на вход как изображение контента, так и векторное представление стиля. Сеть-преобразователь имеет вид кодировщик-декодировщик и имеет нескольких остаточных слоев. В качестве нормализации используется условная нормализация с параметрами вычисленными с помощью векторного представления стиля.

В статье [5] авторы предложили мощную модель, позволяющую производить стилизацию для произвольного стиля. Помимо сети-преобразователя $T(\cdot)$, она имеет также дополнительную сеть - кодировщик стилей (StylePredictor) $S(\cdot)$. Модель работает следующим образом:

1. Кодировщик стилей получает на вход изображение стиля и выдает его векторное представление фиксированной длины
2. Сеть-преобразователь получает на вход вектор стиля.
3. По вектору стиля вычисляются все параметры для условной нормализации.
4. Сеть-преобразователь получает на вход изображение контента и выдает итоговое стилизованное изображение.
5. В сети VGG вычисляются функции потерь и вычисляются градиенты по весам сетей $T(\cdot)$ и $S(\cdot)$

Сеть-преобразователь имеет вид кодировщик-декодировщик, а также имеет 5 остаточных (Residual) слоев в качестве бутылочного горлышка, каждый из которых

состоит из двух сверток и двух условных нормализаций. Таким образом в отличии от AdaIn модель имеет множество блоков отвечающих непосредственно за перенос стиля.

Сеть кодировщик стилей получена из части предобученной сети классификатора InceptionV3 [9] путем добавления блока усреднения по пространственным координатам (Global Average Pooling) и полносвязного слоя. Все веса данной модели является обучаемыми.

3 Улучшение модели с кодировщиком стилей

3.1 Обучение модели

Авторы статьи [5] показали, что, обучая модель более, чем на 16к изображений стиля, можно добиться обощаемости модели на произвольный стиль. Модель имеет около 10 миллионов параметров, и обучить ее на таком большом датасете стилей крайне затруднительно при отсутствии мощных GPU. Поэтому в рамках данной работы было решено упростить модель добавив слой Dropout в конец кодировщика стилей и обучить модель на 5к изображений стиля.

Кроме того, сначала модель обучалась только на 20 фиксированных стилях. После того как модель начала давать качественные результаты на этих же стилях, было решено обучить модель на остальных 5к изображений стиля. Чтобы модель хорошо обучилась но новых стилях, но также продолжила хорошо работать на старых (базовых) был предложен следующий алгоритм обучения модели:

Algorithm 1: Обучение модели на новых стилях.

```
init model NN(·);
init datasets C, S_base, S_new;
init γ;
for (I_c, I_s) ← (C, S_new) do
    ξ = sample from U[0, 1];
    if ξ > γ then
        I_s = sample from S_base;
    end
    I_r = NN(I_c, I_s);
    compute loss, gradients and perform optimizer step;
end
```

Таким образом, на каждой итерации обучения с определенной вероятностью изображение стиля из новой выборки заменятся на базовый стиль. Такой двухэтапный подход к обучению позволяет на основе анализа результатов стилизации на базовых стилях сначала оценить насколько хорошо модель может стилизовать вообще и подобрать параметры весов у функций потерь. Второй этап обучения позволяет дообучить модель на большой выборке стилей, но так чтобы она не «забыла» базовые.

Добавление Dropout, а также новый метод обучения позволил модели обобщаться на новые стили, даже при меньшем размере выборки на обучении по сравнению с экспериментами в исходной статье.

3.2 Контроль силы стилизации

Авторы статьи предложили следующий подход для контроля силы стилизации: сеть кодировщик стилей получает на вход как изображение стиля, так и изображение контента, вычисляет их векторное представление, а дальше выполняет линейную интерполяцию между ними:

$$\hat{s} = \alpha S(I_s) + (1 - \alpha) S(I_c), \tag{1}$$

где $\alpha \in [0, 1]$ - параметр, контролирующий силу стилизации.

Этот подход гораздо проще и эффективнее по сравнению с методом предложенным в статье [6], где для контроля силы стилизации использовалась дополнительная нейросеть, вычислявшая необходимые параметры для условной нормализации.

Однако данный подход также имеет свои недостатки. Во-первых, необходимо чтобы результат интерполяции лежал в распределении векторного представления всех изображений из генеральной совокупности, причем как изображений стиля, так и изображений контента. Другими словами необходимо, чтобы результат интерполяции соответствовал стилю какого-то изображения. Это условие выполняется при обучении модели на датасетах изображений контента и стиля больших размеров, так как в таком случае кодировщик стилей начинает правильно извлекать стилевые паттерны и получать богатое пространство векторных представлений.

Еще одним недостатком является тот факт, что даже при значении параметра α близком к нулю, результат стилизации заметно отличается от изображения контента. Это происходит потому, что в блоках условной нормализации истинные статистики контента заменяются на вычисленные с помощью кодировщика стилей. Чтобы решить эту проблему было решено добавить дополнительные связи из кодировщика в декодировщик, которые затем суммируются с весом равным силе стилизации. Так как основной перенос стиля происходит в остаточных слоях, данные связи позволяют сохранить информацию об искомом изображении контента.

Однако, даже при таком подходе при малых значениях параметра силы стилизации изображени все равно изменяется сетью-преобразователем. Поэтому в рамках практического использования данной модели, как некоторой программы для стилизации, было решено использовать постпроцессинг для решения данной проблемы: при малых α производится интерполяция между исходным изображением контента и стилизованным изображением.

3.3 Контроль силы перекраски

Стилизация изображений часто изменяет цветовую гамму изображения контента. Однако иногда необходимо, чтобы произошел только перенос стилевых паттернов, а сами цвета объектов сохранились. Авторы статьи [7] предложили два подхода, которые позволяют сохранять цвет контента при стилизации изображений. Первый

подход заключатеся в преобразовании изображения из RGB пространства в HCL (hue, chrome and luminance) и стилизации только канала яркости (luminance). Другой подход подразумевает предварительную перекраску изображения стиля в цвета изображения контента. Оба метода показывают примерно равные результаты. Однако второй подход лучше подходит для использования с нашей моделью, а также сохраняет больше информации, так как стилизация происходит по всем каналам. Для перекраски изображений можно использовать метод выравнивания гистограмм. Он заключается в приведении гистограммы интенсивности пикселей изображения стиля к гистограмме контента.

В данной работе этот подход был модифицирован: гистограмма стиля приводится не к гистограмме контента, а к некоторой промежуточной гистограмме, являющейся линейной интерполяцией между гистограммами. Таким образом, мы получаем дополнительный параметр контроля силы перекраски.

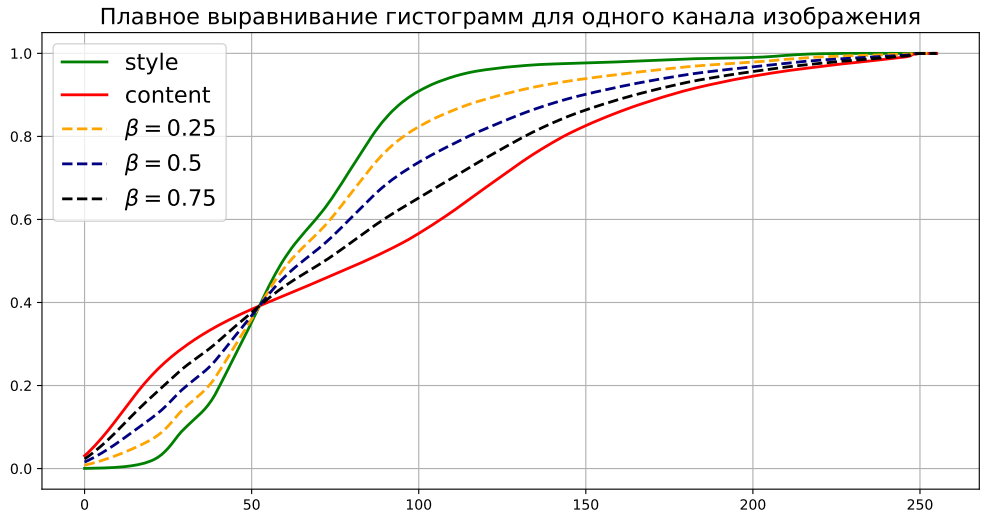


Рис. 2: Визуализация плавного приведения гистограммы стиля к гистограмме контента. Варьируя параметр β можно контролировать силу перекраски.

3.4 Дополнительные ручки контроля стилизации

Так как модель может работать с произвольным стилем, то возможно получить набор векторных представлений различных стилей. Для этого в первую очередь важно, чтобы модель кодировщика стилей правильно выделяла информацию о стиле. В

исходной статье для этого было необходимо обучать модель на большом наборе изображений стиля: более 16к объектов. Однако, в данном случае добавление Dropout помогло сильно упростить модель и получить более равномерное распределение относительно всех компонент векторного представления стилей. Тем самым кодировщик стилей стал правильно выделять информацию о стиле, даже при обучении на 5к объектах.

Анализируя полученное пространство векторных представлений стилей можно увидеть, что существуют ярко выраженные кластеры, заметные даже после преобразования простанства в двумерное с помощью метода главных компонент (PCA). Каждый кластер объединяет в себе объекты имеющие схожие стилевые паттерны.

Так как контроль силы стилизации производится путем интерполяции между векторными представлениями, пользователю предоставляется дополнительная функция: ручки контроля, которые отвечают за веса в линейной интерполяции между центроидами каждого кластера. То есть пользователь, основываясь на примерах изображений стиля из каждого кластера, может самостоятельно задавать какие стилевые паттерны он хочет перенести на изображение контента.

Таким образом, пользователь имеет следующий набор параметров контролирующий стилизацию:

- α - сила стилизации
- β - сила перекраски
- ω - набор весов для сэмплов или центроидов каждого кластера

4 Вычислительные эксперименты

В качестве набора изображений контента использовался датасет MS COCO, состоящий из 118k изображений контента. Для стилей использовалась подвыборка из датасетов PBN (Painter-by-Numbers: 80k изображений стиля - картин) и DTD (Describable Textures Dataset: 5640 изображений текстур). Размер изображений: 360×360 для обучения, 500×500 для теста.

Модель обучалась сначала 20к итераций на 20 базовых стилях. Затем были добавлены дополнительные 5к стилей, на которых модель обучалась уже около 450к

итераций. Также на каждом шаге с определенной вероятностью подмешивались базовые стили.

4.1 Сравнение результатов стилизации

Сравним результат работы нашей модели с бейзлайном, описанным в статье [3]. Бейзлайн может работать с конечным набором фиксированных стилей. Он обучался на базовых стилях нашей модели, а также на некоторой фиксированной тестовой выборке стилей. Далее будем сравнивать работу этих методов на обоих выборках.

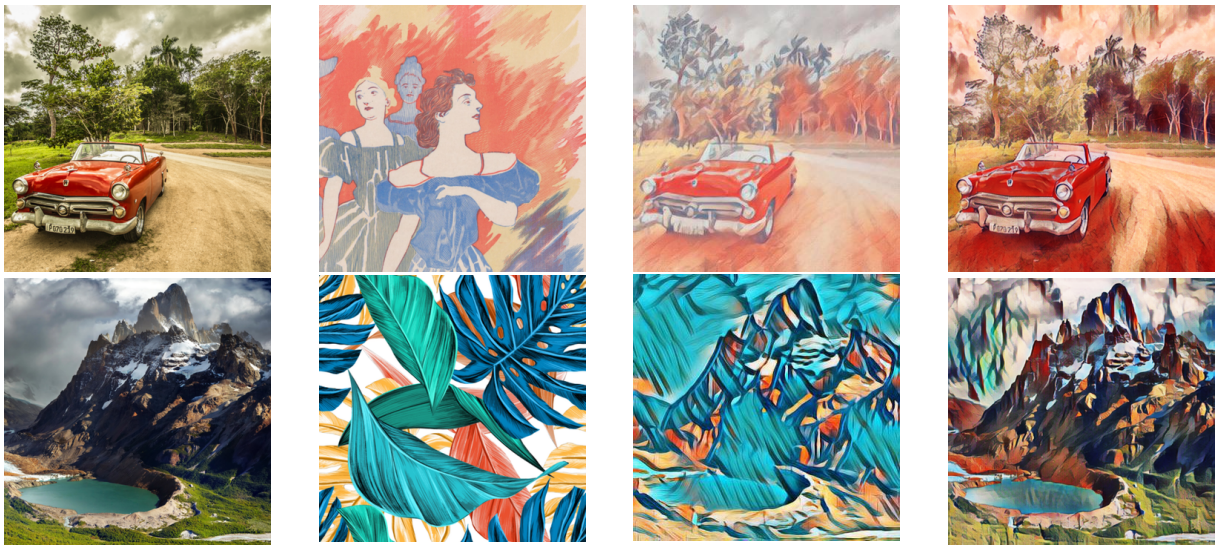
Результат работы метода стилизации на базовых стилях:

Content Style Baseline [3] Ours

Рис. 3: Сравнение результатов стилизации на базовых стилях с бейзлайн моделью.

Видно, что так как модель видела базовые стили во время обучения, она выдает качественную стилизацию сравнимую с оригинальным методом.

Теперь сравним работу метода на тестовых стилях:



Content Style Baseline [3] Ours

Рис. 4: Сравнение результатов стилизации на тестовых стилях с бейзлайн моделью.

Здесь очевидно, бейзлайн работает гораздо лучше, однако достоинство нашей модели заключается в том, хотя эти стили не присутствовали в обучающей выборке, модель выдает некоторую приемлемую стилизацию.

4.2 Оценка контроля силы стилизации

Оценим насколько плавно происходит контроль силы стилизации при изменении параметра α:

Content Style $\alpha = 0.0$ $\alpha = 0.1$ $\alpha = 0.3$ $\alpha = 0.5$ $\alpha = 0.7$ $\alpha = 0.9$ $\alpha = 1.0$

Рис. 5: Сравнение результатов работы метода при разном значении параметра силы стилизации.

Видно, что такой подход к контролю силы стилиазации дает хорошие результаты. Стилизация получается достаточно плавная и пользователь уже на этапе тестирования может самостоятельно задать необходимый для него уровень стилизации.

4.3 Оценка контроля силы перекраски

Оценим насколько изменение параметра силы перекраски влияет на результат стилизации:

Content Style $\beta = 0.0$ $\beta = 0.1$ $\beta = 0.3$ $\beta = 0.5$ $\beta = 0.7$ $\beta = 0.9$ $\beta = 1.0$

Рис. 6: Сравнение результатов работы метода при разном значении параметра силы перекраски.

Видно, что перекраска работает не идеально, но позволяет плавно контролировать изменение цветовой гаммы стилизованного изображения. Важно заметить, что так как бейзлайн модель обучается на фиксированных стилях, то на этапе тестирования для нее невозможен контроль силы перекраски. Если же применять перекраску в качестве постпроцессинга, то изменяется лишь общий тон изображения, а не цвета конкретных объектов.

4.4 Оценка контроля переноса стилевых паттернов

Возьмем некоторую выборку из 20к случайных изображений стиля и преобразуем их в векторное представление. Одно из основных свойств метода кластеризации KMeans - выпуклый вид кластеров. Если после преобразование PCA в двумерное пространство кластеры останутся выпуклыми, то можно судить о том, что пространство действительно хорошо кластеризуемо.

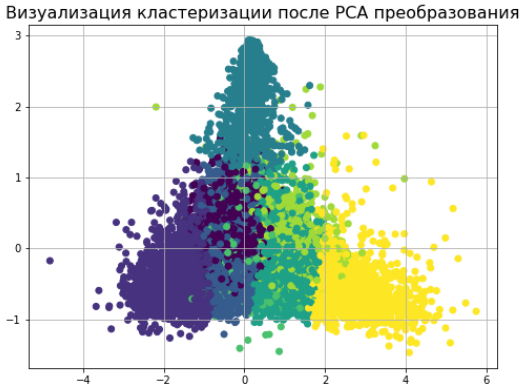


Рис. 7: Визуализация кластеризации в двумерном пространстве. Большинство кластеров имеют выпуклую форму. Каждый кластер соответствует некоторому стилевому паттерну характерному для объектов этого кластера.

Кластеризуем пространство на 8 кластеров и визуализируем центроиды некоторых из них:

Рис. 8: Центроиды полученные после кластеризации стилей в пространстве векторных представлений.

Варьируя веса между центроидами и усредняя векторные представления с этими весами можно получать различные стилизации:

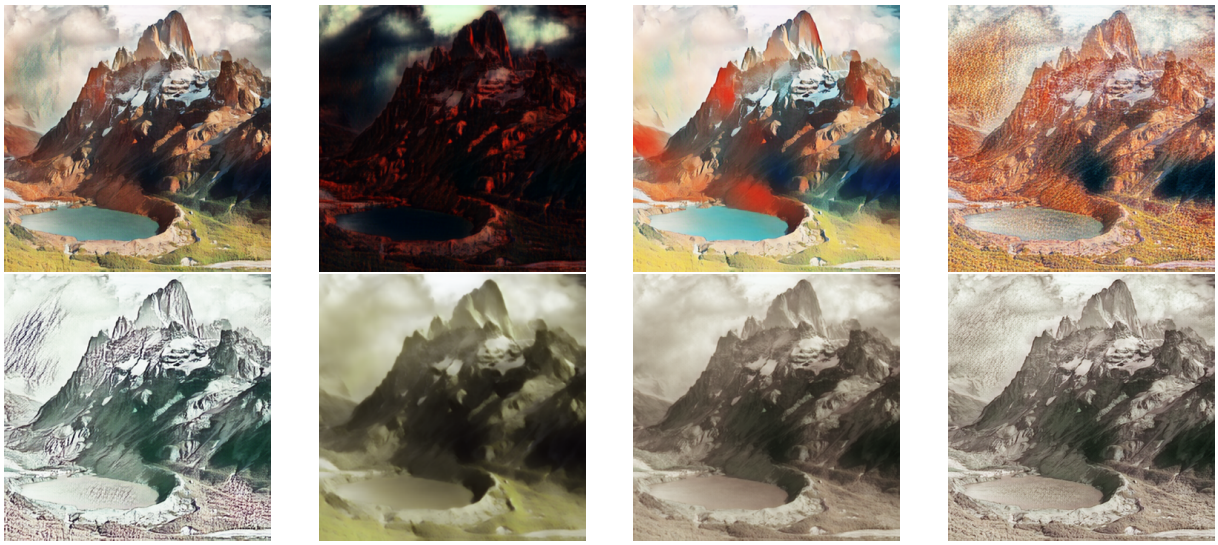


Рис. 9: Различные стилизации со случайными весами у центроид кластеров.

Таким образом, такой подход предоставляет пользователю дополнительную возможность контроля стилизации. Кроме того, он не отвергает возможность контроля силы стилизации. Вообще говоря итоговая формула для контроля стилизации:

$$\hat{s} = \frac{\alpha}{\sum_{i=0}^{n} w_i} \left(S(I_s)w_0 + \sum_{i=1}^{n} s_i w_i \right) + (1-\alpha)S(I_c)$$

5 Заключение

В рамках данной работы выполнено:

- Исследована мощная модель автоматической стилизации изображений: предпосылки ее создания, теоритические и практические аспекты использования, достоинства и недостатки
- Модель с добавлением Dropout обучена на втрое меньшем датасете по сравнению с экспериментами авторов статьи, показана способность модели к работе с произвольным стилем
- Проведена оценка качества работы метода и произведено сравнение с бейзлайн алгоритмом
- Реализованы ручки контроля силы стилизации и силы перекраски, проведена оценка результатов работы

- Реализованы дополнительные параметры контроля стилизации: веса при векторном представлении центроид кластеров пространства стилей, проведена оценка результатов работы

Список литературы

[1] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style."arXiv preprint arXiv:1508.06576 (2015).

[2] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution."European conference on computer vision. Springer, Cham, 2016.

[3] Dumoulin, Vincent, Jonathon Shlens, and Manjunath Kudlur. "A learned representation for artistic style."arXiv preprint arXiv:1610.07629 (2016).

[4] Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization."Proceedings of the IEEE International Conference on Computer Vision. 2017.

[5] Ghiasi, Golnaz, et al. "Exploring the structure of a real-time, arbitrary neural artistic stylization network."arXiv preprint arXiv:1705.06830 (2017).

[6] Babaeizadeh, Mohammad and Ghiasi, Golnaz. "Adjustable Real-time Style Transfer."(2018)

[7] Gatys et al. "Preserving Color in Neural Artistic Style Transfer."(2016)

[8] Simonyan, Karen and Zisserman, Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition."(2014)

[9] Szegedy et al. "Rethinking the Inception Architecture for Computer Vision."(2015)